Week 2

- Using BufferedReader for input
- Using the Scanner class (requires JDK 1.5)
- Variables and Constants
- Primitives and Objects
- String Manipulation
- Arithmetic Operators
- Casting
- Relational and Logical Operators
- Operator precedence

Read Pages 28 – 60
and
Pages 262-270

Sample program Involving Input/Output

write a program to:

- read a string from user

  a prompt should be used

- echo it on screen.

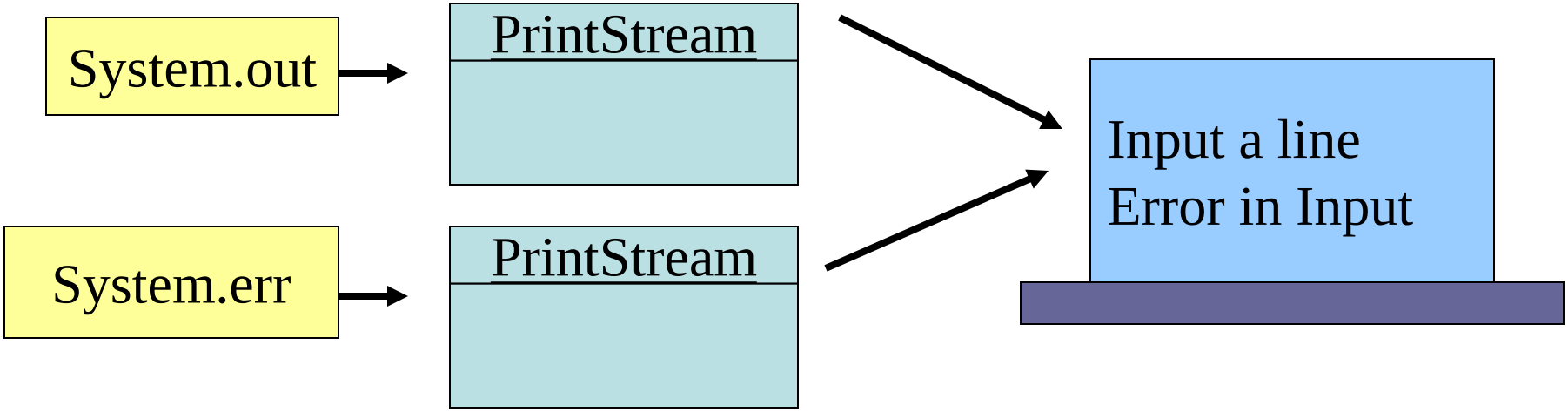
Input a line of text:

my name is Joe Bloggs

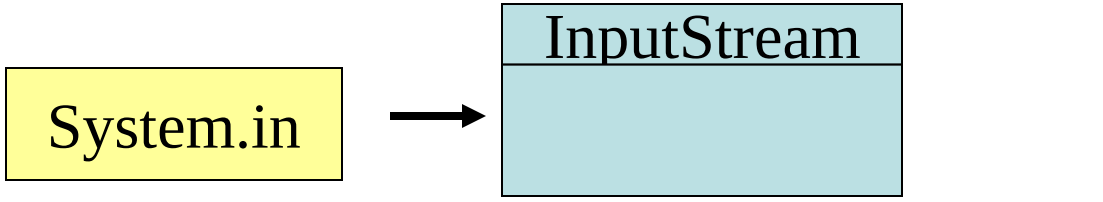
Your input was: my name is Joe Bloggs

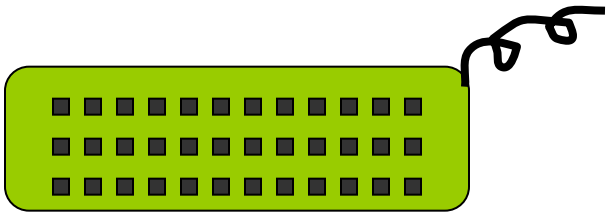
```
import java.io.*;
public class Echo {
   public static void main (String[] args)
         throws IOException {// must include this, more later

      // create a text input stream
      BufferedReader stdin = new BufferedReader
                        (new InputStreamReader (System.in));
      String message;

      System.out.println("Input a line of text");
      message = stdin.readLine();
      System.out.println("Your input was: " + message);
   }
}
```

3 Predefined Objects for Input/Output

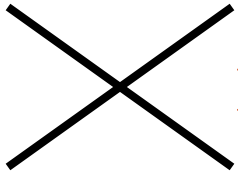



```
System.out.println("Input a line");
System.error.println(”Error in Input");
```





String message = System.in.readln(); ?



No Such method !

InputStream class provides methods for reading bytes.

Java uses Unicode encoding (which allows most world languages to be handled) uses 2 bytes per character.

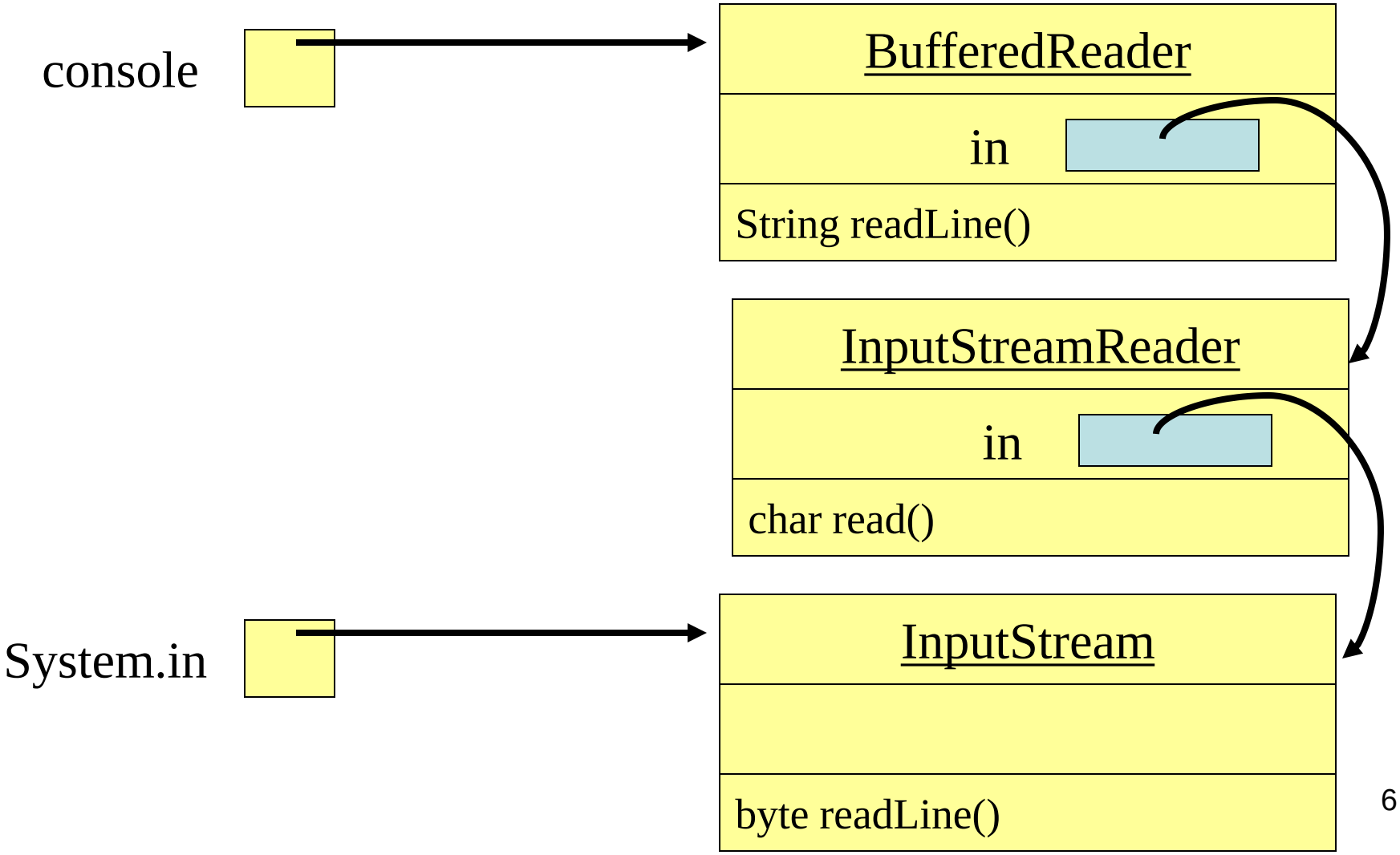
To read characters from System.in we need to chain a InputStreamReader object as in:

```
InputStreamReader reader = new
                        InputStreamReader(System.in);
char ch = reader.read();
```

However to read a line at a time we have to chain it to a BufferedReader object as in:

```
BufferedReader console = new BufferedReader(
                        new InputStreamReader(System.in));
String message = console.readLine();
```

Chaining the objects



Error Handling

- Reading java docs for BufferedReader method readLine() ...

```
public String readLine() throws IOException
```

- This method warns users that it may throw an exception if any errors encountered while reading.
- When a method throws an exception we can either

- Catch the exception and deal with it
- Pass it on by adding *throws IOException* as in:

```
class Echo {
   public static void main (String[] args)
                              throws IOException {
          ...
```

Write a program to add two numbers

- where are the numbers coming from?
- are they whole numbers ? decimal part allowed ?
- where should the output go to?

```
Input the first integer number:
5
Input the second integer number:
7
The sum is:  12
```

```
import java.io.*;
public class AddingInts {
   public static void main (String[] args)
                  throws IOException
   {

      BufferedReader stdin = new BufferedReader
              (new InputStreamReader (System.in));
      String string1, string2;
      int num1, num2, sum;

      System.out.println ("Input an integer number");
      string1 = stdin.readLine();
      num1 = Integer.parseInt (string1);

      System.out.println ("Input another integer number");
      string2 = stdin.readLine();
      num2 = Integer.parseInt (string2);

      sum = num1 + num2;
      System.out.print("The sum is: " + sum);
      System.out.println();
   }
}
```

So, what's new ?

- To store whole numbers use integer variables as in:

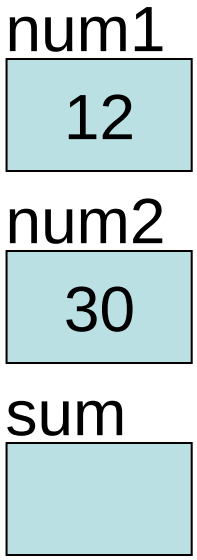
```
int num1, num2, sum;
```

- To add the values stored in num1 and num2 and to store it in sum use an assignment statement as in:

```
sum = num1 + num2;
```

- The use of + with a String and an int as in:

```
System.out.println("The sum is: " + sum);
```



How did we read an integer ?

- Create and chain the BufferedReader object to System.in

```
BufferedReader stdin = new BufferedReader
          (new InputStreamReader (System.in));
```

- Read and store the user input in a String object

```
String string1;
string1 = stdin.readLine();
```

- Be prepared to handle the exception thrown by readLine()

```
public static void main(String[] args)throws IOException
```

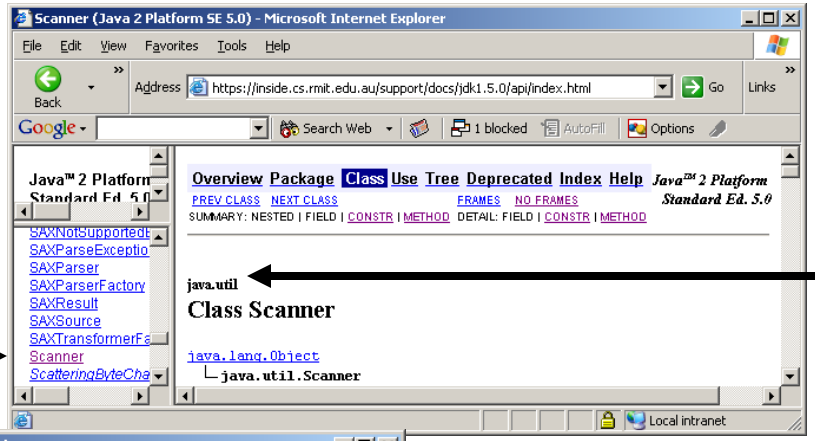
- To convert a String to integer use Integer.parseInt(.) as in

```
num1 = Integer.parseInt (string1);
```

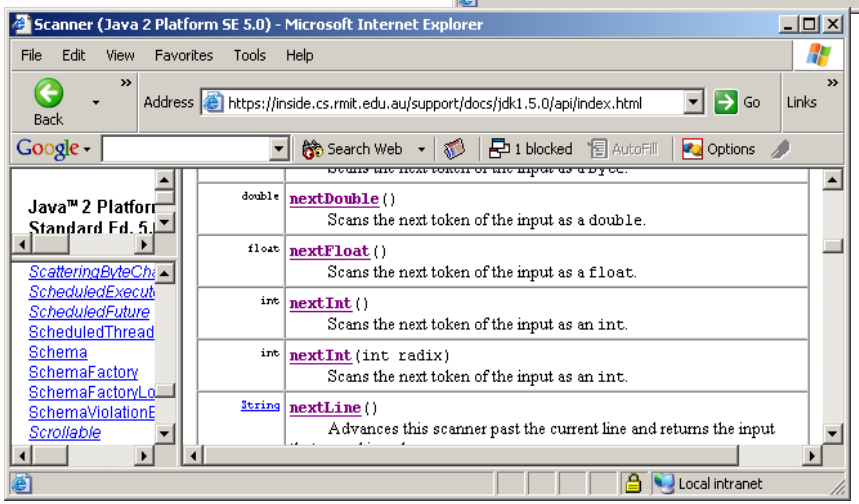
Too tedious to read. Any easier way ?
JDK 1.5 comes to the rescue

Checking out the Scanner class

Refer to the Scanner in Java Docs



In the package java.util



Methods of interest that read and return the next int, double and return

Using the Scanner class

```
import java.util.*;
public class Echo {
   public static void main (String[] args)  {
      Scanner console = new Scanner(System.in);
      System.out.println("Input a line of text");
      String message = console.nextLine();
      System.out.println("Your input was: "
                  +  message);
   }
}
```

Import the packages

- Call the nextLine() method on the Scanner object to read and return a string
- Similarly call nextInt() and nextDouble() to read and return int and double values.

To constructor a Scanner object pass the predefined reference System.in

2nd program rewritten using Scanner

```
import java.util.*;
public class AddingInts {
   public static void main (String[] args) {
      Scanner console=new Scanner(System.in);
      int num1, num2, sum;

      System.out.println ("Input an integer number");
      num1 = console.nextInt();

      System.out.println ("Input another integer number");
      num2 = console.nextInt();

      sum = num1 + num2;
      System.out.println("The sum is: " + sum);
   }
}
```

No Chaining

No exception handling

No Strings required

No conversion to ints required

All details handled by Scanner class

Quiz Time

Q1. How do you read an integer/double value using scanner? _____

Q2. Complete the program below to read a number and to print its square using only one variable.

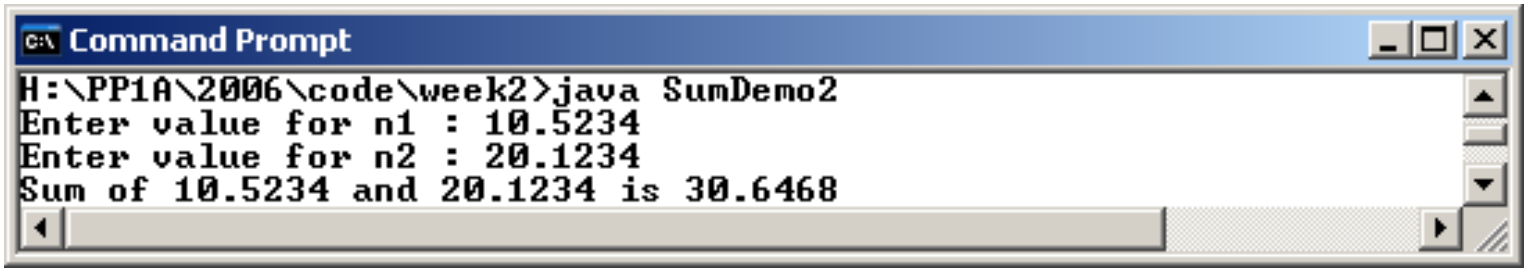
```
import java.util.*;
public class VarTest1
{
   public static void main(String args[])
   {
      …


   }
}
```

Expected Input/Output (input underlined)

Enter a number : 6
Square of that number = 36

Output using print()

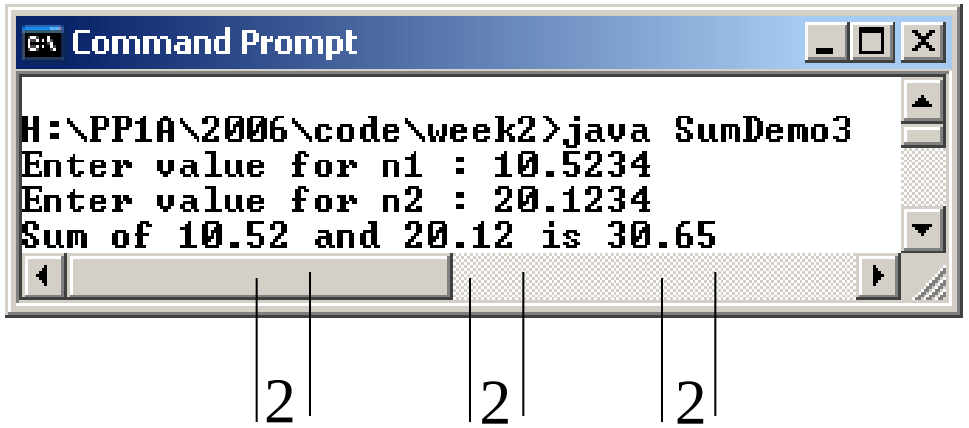
```
import java.util.Scanner;
public class SumDemo2 {
   public static void main (String[] args)
   {  Scanner keyboard = new Scanner(System.in);
      System.out.print("Enter value for n1 : ");
      double n1 = keyboard.nextDouble();
      System.out.print("Enter value for n2 : ");
      double n2 = keyboard.nextDouble();
      double sum = n1 + n2;
      System.out.print("Sum of "+n1+ " and " + n2
           + " is " + sum);
   }
}
```



Unlike print(), printf() allows setting width & decimal places directly.

Formatted output using printf()

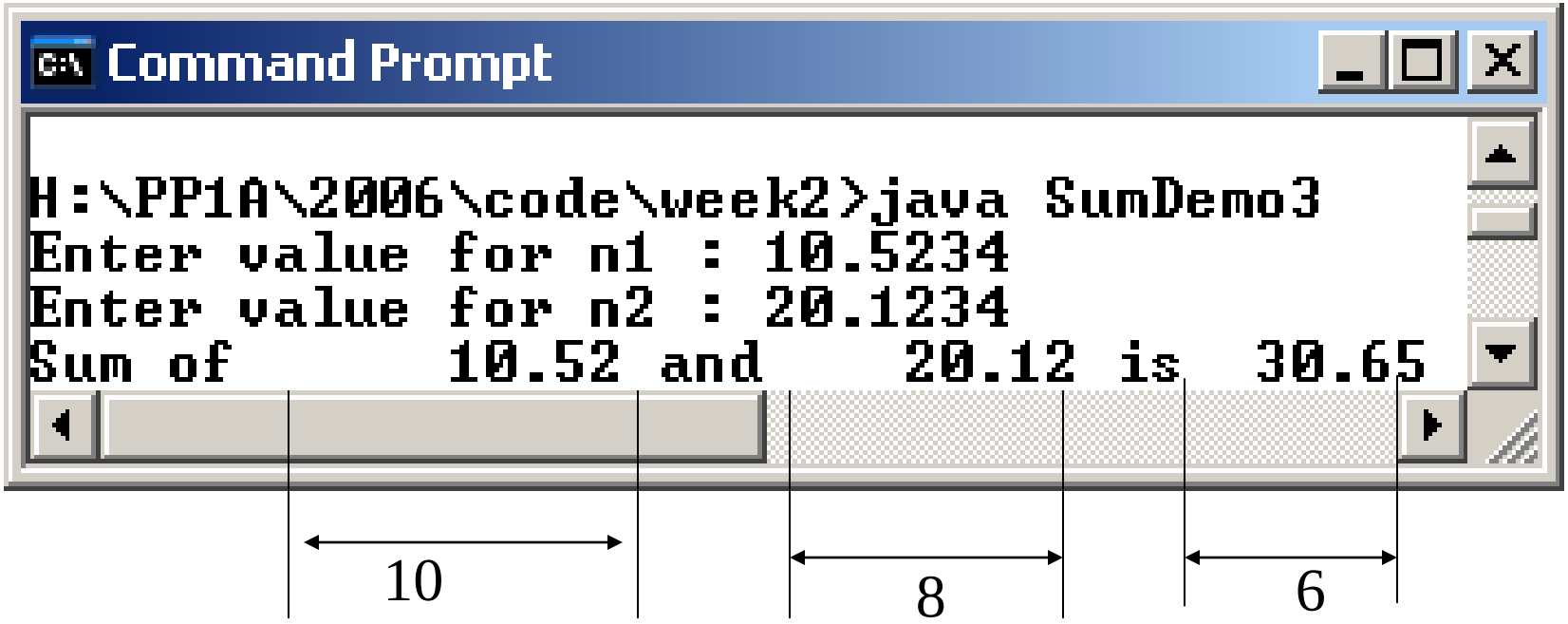
```
import java.util.Scanner;
public class SumDemo3 {
   public static void main (String[] args)
   {  Scanner keyboard = new Scanner(System.in);
      System.out.print("Enter value for n1 : ");
      double n1 = keyboard.nextDouble();
      System.out.print("Enter value for n2 : ");
      double n2 = keyboard.nextDouble();
      double sum = n1 + n2;
      System.out.printf("Sum of %.2f and %.2f is %.2f"
                        ,n1,n2,sum);
   }
}
```



What if I change to specify field width?

In the previous program changing last statement to:

```
System.out.printf("Sum of %10.2f and %8.2f is %6.2f"
                  ,n1,n2,sum);
```



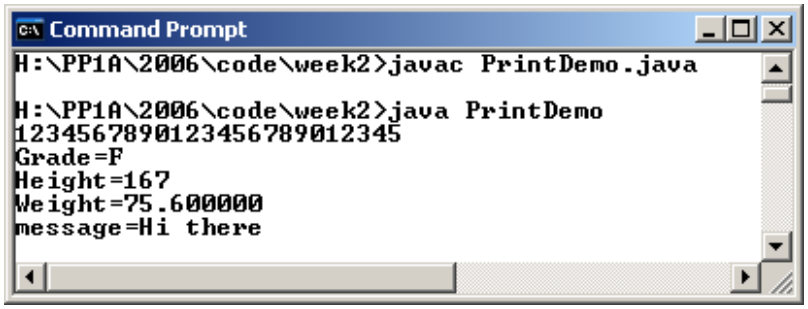
Formatting output with printf()

- %c a character
- %d a decimal intger
- %f a floating point number
- %s a string

Using the printf()

JDK 1.5 has added printf allowing formatted output

```
public class PrintDemo {
    public static void main (String[] args)
    {
        int ht = 167;
        String message = "Hi there";
        double wt = 75.60;
        char grade = 'F';
        System.out.println("123456789012345678901234");
        System.out.printf("Grade=%c\n",grade);
        System.out.printf("Height=%d\n",ht);
        System.out.printf("Weight=%f\n",wt);
        System.out.printf("message=%s\n",message);
    }
}
```



Coding Style

- Class Names start with a capital letter
- If there are several words start each word with uppercase letter

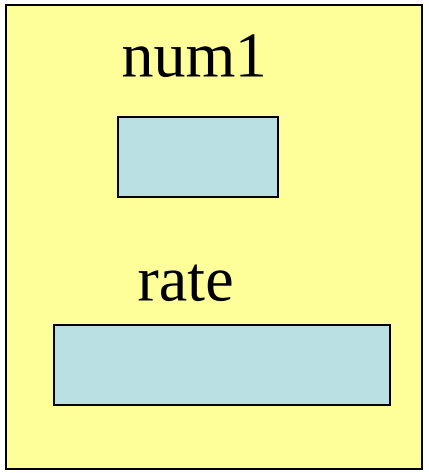
  Echo, AddingInts, HelloWorld
- For variables start with a lowercase letter

  int anIntegerColor;

  String employeeName;

Variables and Constants

- Variables are used for storing values that change over the course of program execution.

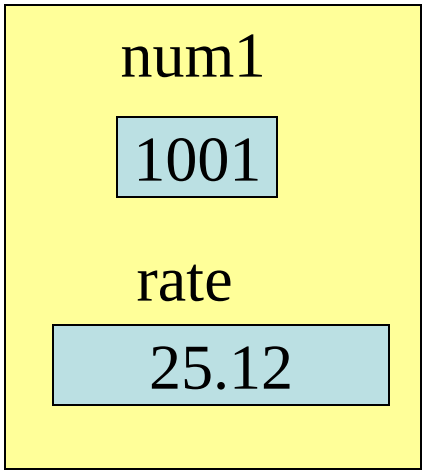


- How do we instruct the compiler we need a variable (declare) ?

```
int num1;     // reserves 4 bytes
double rate; // reserves 8 bytes
var_type varName; // general format
```

- Can we initialize at declaration ? Yes.

```
int num1 = 1001;
double rate = 12.56 * 2;
// general format
var_type varName = expression;
```



Variables and Constants

What if I want the value to remain fixed ?

```
final int MAX_STUDENTS = 100;
final int TAX_BRACKET = 30000;
```

Use all uppercase letters for constants

The keyword final has different meanings when applied to different things such as classes and methods.

Quiz: Which of the following are valid statements ?

(a) final double PI = 3.14;

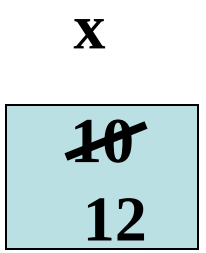
(b)
```
final PI;
PI = 3.14;
```

(c)
```
final double PI = 3.1;
PI = 3.14;
```

Primitives and Objects

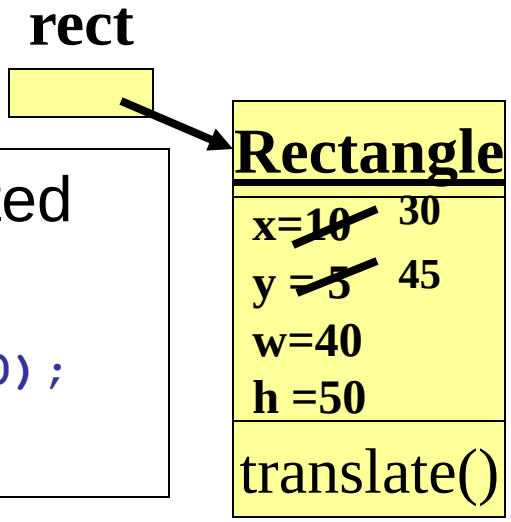
Primitives such as ints and doubles are created and manipulated directly using their names. (Though compiler uses their addresses it does not concern us)

```
int x = 10;

x = x + 2;
```



Objects have no names and are manipulated indirectly using an object reference.

```
Rectangle rect = new Rectangle(10,5,40,50);

rect.translate(20,40);
```



Quiz 1

What will be the output of the program below? Trace!!!

```
public class TestVar
{  public static void main (String[] args)
   {
      int x = 10;
      int y = 20;
      System.out.println("x & y are now " + x + " & " + y);
      y = x + 2;
      System.out.println("x & y are now " + x + " & " + y);
      x = y + 2;
      System.out.println("x & y are now " + x + " & " + y);
      y = x + 2;
      System.out.println("x & y are now " + x + " & " + y);
   }
}
```

Ans: _____

_____

_____

_____

Quiz 2

What will be the output ?

```
public class TestVar1
{  public static void main (String[] args)
   {
      int x = 10;
      x = x + x;
      x = x + x;
      x = x + x;
      System.out.println("x is now " + x);
   }
}
```

Ans: _____

Quiz 3

What will be the output ?

```
public class TestVar2
{
   public static void main (String[] args)
   {
      int amount = 356;
      System.out.println("amount is now " + amount);
      amount = amount % 200;
      System.out.println("amount is now " + amount);
      amount = amount % 100;
      System.out.println("amount is now " + amount);
      amount = amount % 50;
   }
}
```

Ans: _____

_____

_____

_____

Quiz 4

Program below is trying to swap num1 & num2. What will be the output?

```
public class Swap {
   public static void main (String[] args)
   {
      int num1 = 10, num2 = 20;
      num1 = num2;
      num2 = num1;
      System.out.println("num1="+num1+" num2="+num2);
   }
}
```

Ans: _____

Quiz 5

Modify the program to swap num1 and num2 correctly (Hint use a third int variable, say temp)

```
public class SwapCorrect {
   public static void main (String[] args)
   {
      int num1 = 10, num2 = 20;
      int temp;
      …



      System.out.println("num1="+num1+" num2="+ num2);
   }
}
```

Primitive Data Types

- Primitives in Java are byte, short, int, float, double, char and boolean. The first 4 are numeric types.
- Some problems require us to handle whole numbers while others require floating point numbers (with decimal part).
- What kind of variable would you use for:

  day, month, year ?

  Temperature, weight(kg) ?

Integers and Floating Point

- Integer numbers (using byte, short and int) are stored using exact representation
- Floating point numbers are stored separately using the mantissa and exponent.As the number of significant digits that can be stored is limited they cannot be represented exactly. Hence Floating point representations are only approximations.

$27.56 = 2.756 * 10^1$ mantissa = 2756 exponent = 1

- Most high-level languages don’t specify the sizes of their data types - impediment for portability.
- Java specifies the sizes precisely (independent of platform)

Java Numeric Types

Type	*Size*	*Minimum*	*Maximum*
`byte`	8 bits	–128	127
`short`	16 bits	–32,768	32,767
`int`	32 bits	–2,147,483,648	2,147,483,647
`long`	64 bits	–9,223,372,036,854,775,808	9,223,372,036,854,775,807
`float`	32 bits	App. –3.4E+38	App. +3.4E+38
`double`	64 bits	App. –1.7E+308	App. +1.7E+308

- Typically use int for integers and double for floating point
- For large integers use long
- For small integers (day,month,year) you may use byte if memory is a constraint

Data Types char and boolean

- Most languages such as C, Pascal use 1 byte for char (8 bits) allowing only 256 distinct characters to be stored.
- Java uses 2 bytes (Unicode) for characters allowing asian languages to be stored (up to 65536 values)
- But most Operating Systems use 1 byte character - hence we have to convert them using classes such as InputStreamReader.

```
char input = 'q';

char ch = '\n';  // \n - newline  and  \t - tab
```

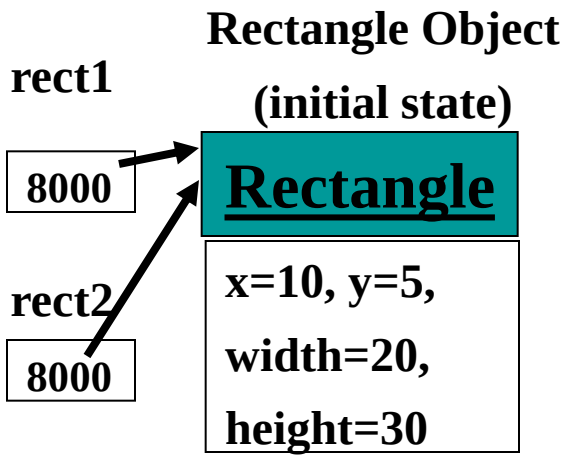
- boolean is used for storing true / false values

```
boolean stop = false;
```

Manipulating Objects through references

What will be the output?

```
import java.awt.Rectangle;
public class HelloRectangle  {
   public static void main(String[] args){
      Rectangle rect1 = new
                  Rectangle(10,5,20,30);
      Rectangle rect2 = rect1;
      rect1.translate(20,30);  // shifting
      rect2.translate(10,20);
      System.out.println(rect1);
   }
}
```



Output from the program

```
java.awt.Rectangle[x =__ , y =__ , width = 20, height = 30]
```

Strings

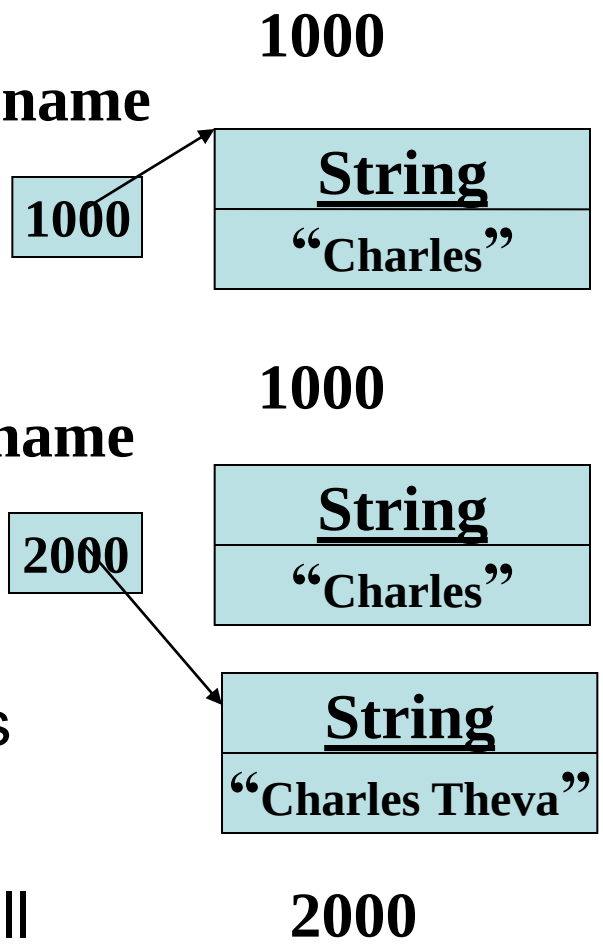
```
String name = new String("Charles");
```

is equivalent to: (strings are special)

```
String name = "Charles";
```

What if I write the statement below next ?

```
name = "Charles Theva";
```



A new String object will be created and it's address will be assigned to name.

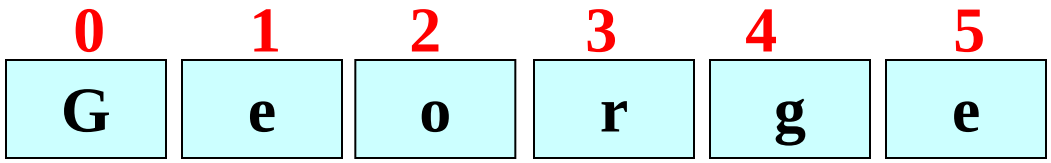
The old String object (at address 1000) will be marked for garbage collection.

Once a String object is created it cannot be changed (immutable object) in any way!

String Operations

```
String myName = "George";
```

the snapshot of memory is



Note that the first index is 0, not 1. Once a String object is created we can use its methods, such as

```
myName.charAt(2);     // returns 'o'
myName.indexOf('e'); // returns 1, it is the first 'e'
myName.indexOf("or");// returns 2 as "or" starts at 2
```

If mutable strings are required StringBuffer class should be used as in:

```
StringBuffer name = new StringBuffer("Cherles");
name.setCharAt(2,'a'); // "Cherles " -> Charles
```

Quiz

Program demonstrates String operations length() and charAt().

```
import java.util.*;
public class TestString
{  public static void main(String args[])
   {  String s;
      Scanner console = new Scanner(System.in);
      System.out.print("Enter a String : ");
      s = console.nextLine();
      int len = s.length();
      System.out.println("Length of s1 is " + len);
      System.out.print("Specify index of character 0 .. " + (len-1)+ " : ");
      int index = console.nextInt();
      System.out.println("char at index "+ index+" is "+  s.charAt(index));
   }
}
```

(a) What will be the output if input String is Elephant and index is 1 ?

(b) input String is Cat and index is 3 ?

(i) What is the name of Exception thrown (if any) ? _____

(ii) Why was the exception thrown? _____

String operations

String concatenation

The concat() method or the operator + to concatenate two strings as in:

```
String s = s1.concat(s2);
String s = s1 + s2;
```

Note that the operator + can be used to concatenate a string with a number.

Quiz

What will the output of the program below?

```
import java.util.*;
public class StringConcat
{
   public static void main(String args[])
   {
      String s1 = " The Truth";
      String s2 = " The Way";
      String s3 = " and The Life";
      String s4 = s1 + s2 + s3;
      System.out.println(s4);
      System.out.println(s1 + s2);
     System.out.println(1 + s1);
   }
}
```

Ans: _____

_____

_____

SubStrings

The substring() method creates a substring. It has two forms:

```
public String substring(int startIndex, int endIndex)
public String substring(int startIndex)
```

The first one returns a new string formed by characters from startIndex to endIndex-1.

The second one returns a new string formed by characters from startIndex to the end of string.

Quiz

What will the output of the program below?

```
public class StringConcat
{
   public static void main(String args[])
   {
      String s1 = " The Truth";
      String s2 = " The Way";
      String s3 = " and The Life";
      String s4 = s1 + s2 + s3;
      System.out.println(s4);
      System.out.println(s1 + s2);
     System.out.println(1 + s1);
   }
}
```

Ans: _____

_____

_____

Quiz

What will the output of the program below?

```
public class StringSub1
{ public static void main(String args[])
   {  String s1 = "Chan Kim Loy";
      String s2 = s1.substring(5);
      String s3 = s1.substring(5,7);
      System.out.println(s2);
      System.out.println(s3);
    }
}
```

Ans: _____

_____

What will the output if we add the statement below at the end?

```
System.out.println(s1.substring(0,50));
```

Ans: _____

String Conversions

- Recall contents of an existing String object cannot be changed as it is immutable.
- However it provides a number of useful methods that creates a new string based on existing one.
- toLowerCase() – returns a string with all lowercase
- toUpperCase() – returns a string with all uppercase
- replace(char c1, char c2) – returns a String with all char c1 replaced with c2.
- replaceAll(String s1, String s2) – returns a String with all char s1 replaced with s2.

Quiz

What will the output of the program below?

```
public class StringConcat
{
   public static void main(String args[])
   {
      String s1 = " The Truth";
      String s2 = " The Way";
      String s3 = " and The Life";
      String s4 = s1 + s2 + s3;
      System.out.println(s4);
      System.out.println(s1 + s2);
     System.out.println(1 + s1);
   }
}
```

Ans: _____

_____

_____

Quiz

What will the output of the program below?

```
public class StringSub2
{ public static void main(String args[])
   {
      String s1 = "Haste makes waste";
      String s2 = s1.toLowerCase();
      String s3 = s1.toUpperCase();
      String s4 = s1.replace('a','e');
      String s5 = s1.replaceAll("aste","ill");
      System.out.println(s2);
      System.out.println(s3);
      System.out.println(s4);
      System.out.println(s5);
   }
}
```

Ans:

_____

_____

_____

_____

Comparing Strings

- String contents cannot be compared directly using s1 == s2 (compares references). Instead use compareTo().
- This method compares character by character, until one character in s1 is less than its corresponding character in s2, then s1.compareTo (s2) < 0 is true
- if s1 is shorter than s2, and the characters in s1 are identical to the corresponding characters in s2, then s1.compareTo (s2) < 0 is true

Examples:

```
"George".compareTo ("George") == 0  is true
"Georg".compareTo ("George")   < 0  is true
"George".compareTo ("Georg")   > 0  is true
"GEORGE".compareTo ("George")  < 0  is true
"   pam".compareTo ("pam")     < 0  is true
```

Operators

Arithmetic Operators + - * / %

What will be the output ?

```
int  x = 10;
x = x + 5;       // short form x += 5;
x = x / 4;       // short form   x /= 4;
x = x % 2;       // short form   x %= 2;
System.out.println("Current value of x is "+x);
```

Quiz

What will be the program output with input specified below. Explain why. (a) deer & dog (b) cat & cats (c) Cat & cat

```
import java.util.*;
public class CompareString
{   public static void main(String args[])
    {  String s1, s2;
       Scanner console = new Scanner(System.in);
       System.out.print("Enter 1st String : ");
       s1 = console.nextLine();
       System.out.print("Enter 2nd String : ");
       s2 = console.nextLine();
       if ( s1.compareTo(s2) > 0 )
           System.out.println( s1 + " > " + s2);
       if ( s1.compareTo(s2) == 0 )
           System.out.println( s1 + " == " + s2);
       if ( s1.compareTo(s2) < 0 )
           System.out.println( s1 + " < " + s2);
     }
}
```

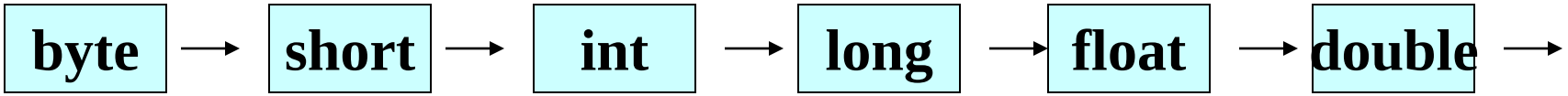
Ans: _____

_____

_____

Numeric Type Conversions

In Java the range of numeric types increase in this order.



A value with a given type can be assigned to variable of any other type which has a greater range.

```
int x = 100;
double y = x;
long z = x;
```

→ No problem

→ No problem

However a value cannot be assigned to a variable with a smaller range, without explicit casting. This prevents possible overflow.

```
double y = 99.9;
int x = y;
int y = (int) y;
```

→ Compilation Error

→ No problem

When variables of different types are used in an expression the variables are automatically converted to the larger type.

```
10 * 30.0
```

→ Expression evaluates to double value

Casting

What will be the output of program segment below? Why?

```
double ratio1, ratio2, ratio3;
int x = 10;
int y = 20;
ratio1 = x/y;
ratio2 = (double) (x / y);
ratio3 = (double) x / y;
System.out.println(" 1. ratio1 = " + ratio1 );
System.out.println(" 2. ratio2 = " + ratio2 );
System.out.println(" 3. ratio3 = " + ratio3 );
```

Ans: _____

_____

_____

Relational and Logical Operators

- The relational operators <, <=, >, >= have the natural meaning yielding a boolean value(true/false).
- To further manipulate these values Java provides the logical operators && (AND), || (OR), ^ (XOR), and ! (NOT).
- These logical operators can be applied to boolean (true/false) operands only. The result of these operations are given in the truth table below.

a	b	!a	a && b	a||b	a^b
T	T	F	T	T	F
T	F	F	F	T	T
F	T	T	F	T	T
F	F	T	F	F	F

What will be the output ?

```
public class Ops
{
   public static void main(String[] args)
   {
       int x = 10, y = 20, z = 15;
       boolean check1, check2, check3, check4;

       check1 = y > x;
       check2 = y > x && x > y;
       check3 = y > x || x > y;
       check4 = y > x && !(x > y);
       System.out.println("check1 = " +check1);
       System.out.println("check2 = " +check2);
       System.out.println("check3 = " +check3);
       System.out.println("check4 = " +check4);

   }
}
```

Ans: _____

_____

_____

_____

Shorthand Operators

- The statement `x = x + 5;` uses two separate operators + (higher precedence) and = to add 5 to x and to store the result in same variable x.

- Short hand operators such as +=, -=, *=, /=, %= allow corresponding operations to be combined.

- For example, using the addition assignment operator += we can create an equivalent statement:

```
x += 5;   // equivalent to x = x += 5;
```

Post/Pre Increment/Decrement Operators

- Operators ++ and -- allow incrementing/decrementing by 1.
- If used in prefix form, expression uses the altered value of variable. In the code below expression ++x * 2 evaluates to 22 (11*2). Hence y is set to 22.

```
int  x = 10;
int y = ++x * 2;
System.out.print(" x= " +x + " y= " + y);
```

output

x=11 y=22

- In postfix form, expression uses the original value of variable. In the code below, expression x++ * 2 evaluates to 20 (10 * 2). Hence y is set to 20.

```
int  x = 10;
int y = x++ * 2;
System.out.print(" x= " +x + " y= " + y);
```

output

x=11 y=20

- Note in both cases the value stored in variable x is incremented by 1.
- Similarly -- can be used in pre or post form.

Quiz on pre/post increment/decrement

What will be the output ?

```
public class Ops
{    public static void main(String[] args)
      {
       int a = 10,p,q,r,s;
       p = a++; // assign a to p then add 1
       q = ++a; // add one to a then assign to p
       r = --a;
       s = a--;
       System.out.println("p  = " + p);
       System.out.println("q = " + q);
       System.out.println("r = " + r);
       System.out.println("s = " + s);
       System.out.println(”a = " + a);
      }
}
```

Ans: _____

_____

_____

_____

_____

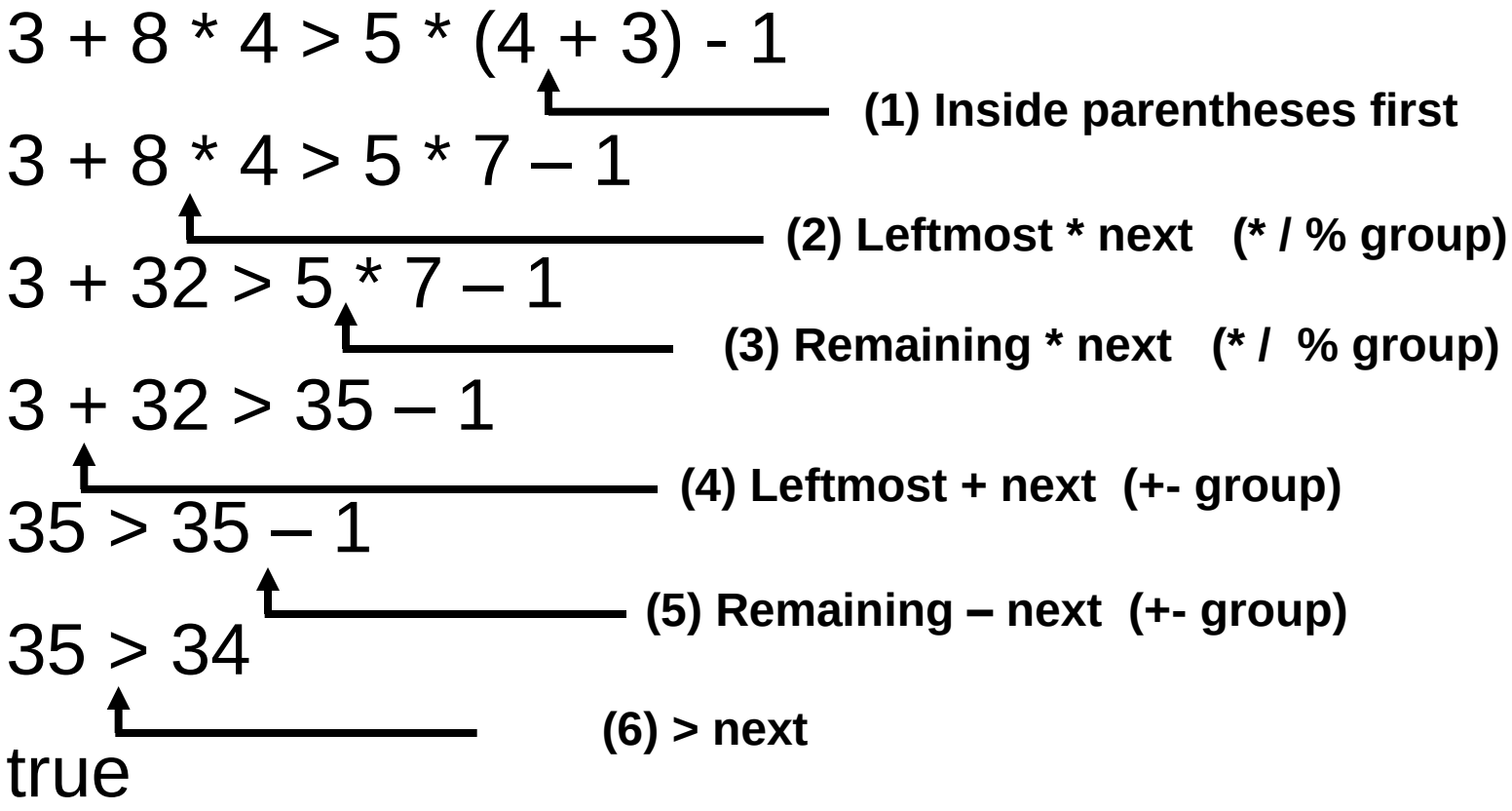
Operator precedence & Associative Rule

- Order in which operations are evaluated are based on precedence chart (below) and associative rule.
- Associative Rule: Operators with equal precedence are carried out from left to right, except for = operator (which is right to left)

Category	***Operators***	***Associativity***
Operations on references	`. []`	L to R
Unary	`++ -- ! - (type)`	R to L
Multiplicative	`* / %`	L to R
Additive	`+ -`	L to R
Shift (bitwise)	`<< >> >>>`	L to R
Relational	`< <= > >= instanceof`	L to R
Equality	`== !=`	L to R
Boolean (or bitwise) AND	`&`	L to R
Boolean (or bitwise) XOR	`^`	L to R
Boolean (or bitwise) OR	`|`	L to R
Logical AND	`&&`	L to R
Logical OR	`||`	L to R
Conditional	`?:`	R to L
Assignment	`= *= /= %= += -=`	R to L

Order of Evaluation:

Expression 3 + 8 * 4 > 5 * (4 + 3) - 1



Quiz: What will be the output of program below.

```
public class PrecedenceTest
{
   public static void main(String args[])
   {
      int x = 30;
      int y = 2;
      int z = 3;
      int a = x / y+z;
      int b = x / y*z;
      int c = x * y+z;
      int d = x / (y+z);
      System.out.println("a="+a+" b="+b+" c="+c+" d="+d);
   }
}
```

Ans: _____

Quiz: State the results for valid expressions below

Valid Expressions	Results
8 > 5	true
12 > 18 && 24>12	…
12>18 || 24>12	…
12>18 ^ 24>12	…
18>12 ^ 24>12	…
8 > 7 == 9 > 4	…

Quiz: State the reason why expression below are invalid

Invalid Expressions	State why expression invalid
8 && 5	
10 > 8 > 5	
! 10 > 8	
5 || 8 < 10	
8+3 = 7+4	
true > false	

Identify the problem with this program attempting to find the average & correct it!

```
// Trying to find the average of two numbers
public class Ops
{
   public static void main(String[] args)
   {
      double a = 10.0;
      double b = 20.0;

       double aver = a + b  / 2;
       System.out.println("average of  a and b  = "+aver);
   }
}
```